ON THE INADEQUACY OF CKA AS A MEASURE OF SIMILARITY IN DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Comparing learned representations is a challenging problem which has been approached in different ways. The CKA similarity metric, particularly it's linear variant, has recently become a popular approach and has been widely used to compare representations of a network's different layers, of similar networks trained differently, or of models with different architectures trained on the same data. CKA results have been used to make a wide variety of claims about similarity and dissimilarity of these various representations. In this work we investigate several weaknesses of the CKA similarity metric, demonstrating situations in which it gives unexpected or unintuitive results. We then study approaches for modifying representations to maintain functional behavior while changing the CKA value. Indeed we illustrate in some cases the CKA value can be heavily manipulated without substantial changes to the functional behaviour.

1 INTRODUCTION

In the last decade, increasingly complex deep learning models have taken over machine learning and have helped us solve, with remarkable accuracy, a multitude of tasks across a wide array of domains. Due to the size and flexibility of these models it has been particularly hard to study and understand exactly *how* they solve the tasks we use them on. A helpful framework for thinking about these models is that of *representation learning*, where we view artificial neural networks (ANNs) as learning internal representations of the input data that are progressively more expressive in terms of the input's semantic information as we go deeper into the ANN layers. In practice, it is often of interest to analyze and compare the representations of multiple ANNs, however, the typically high dimensionality of ANN internal representation spaces makes this a fundamentally difficult task.

To address this problem, the machine learning community has tried finding meaningful ways to compare ANN internal representations and various *representation (dis)similarity measures* have been proposed. Recently, Centered Kernel Alignment (CKA) (Kornblith et al., 2019) was proposed and shown to perform significantly better on representation similarity "sanity checks" than past methods such as linear regression or CCA based methods (Raghu et al., 2017; Morcos et al., 2018). While CKA can capture different notions of similarity between points in representation space by using different kernel functions, it was empirically shown in the original paper that there are no real benefits in using CKA with a non-linear kernel over it's linear counterpart. As a result, linear CKA has been the preferred representation similarity measure of the machine learning community in the past three years and all other similarity measures have been, for all practical purposes, retired. CKA has been utilized in a number of works to make conclusions regarding the similarity between different models and their behaviours such as wide versus deep ANNs (Nguyen et al., 2021) and transformer versus CNN based ANNs (Raghu et al., 2021). They have also been used to draw conclusions about transfer learning (Neyshabur et al., 2020) and catastrophic forgetting (Ramasesh et al., 2021).

However, very recent work has questioned CKA's supremacy as a representation similarity measure by highlighting some of its shortcomings. In particular, Ding et al. (2021) demonstrated that CKA lacks *sensitivity* to changes that affect functional behaviour being unable to detect removal of principal components from the analyzed representations even when this removal significantly decreases probing accuracy. Williams et al. (2021) discussed how CKA does not respect the triangle inequality, which makes it problematic to use CKA values as a similarity measure in downstream analysis tasks. In this work, we further explore the inadequacies of linear CKA as a similarity measure in deep learning and showcase scenarios in which it behaves in undesirable ways, giving either high similarity measures in scenarios where the similarity should be low or the opposite. We also present cases where the CKA value can be heavily manipulated to be either high or low without significant changes to the functional behaviour of the underlying ANNs.

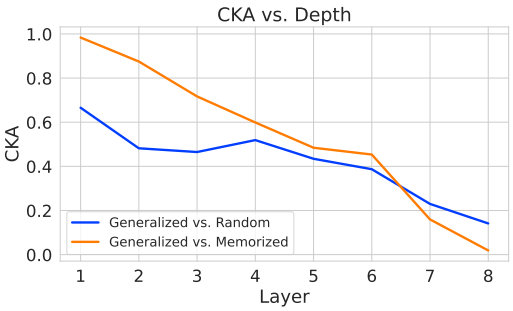

Figure 1: A layer-wise comparison based on the value of the CKA between a generalized, memorized, and randomly populated network. This comparison reveals that early layers of these networks achieve relatively high CKA values.

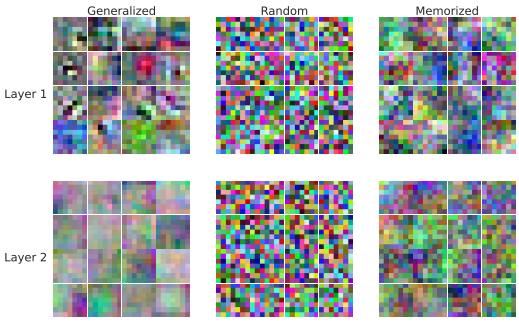

Figure 2: The convolution filters within the first two layers of a generalized, memorized, and a randomly initialized network elucidates that the features are (1) drastically different, and (2) not equally useful despite the CKA results in Fig. 1

2 Problem Statement and Background on CKA

Let $X \in \mathbb{R}^{n \times d_1}$ denote a set of ANN internal representations, i.e. the neural activations of a specific layer with d_1 neurons in a network, in response to $n \in \mathbb{N}$ input examples. Let $Y \in \mathbb{R}^{n \times d_2}$ be another set of such representations generated by the same input examples but possibly at a different layer of the same, or different, deep learning model. It is standard practice to center these representations column-wise (feature or "neuron" wise) before analyzing them. We are interested in representation similarity measures which try to capture a certain notion of similarity between X and Y. CKA is one such similarity measure based on the Hilbert-Schmidt Independence Criterion (HSIC) (Gretton et al., 2005) that was presented as a means to evaluate independence between random variables in a non-parametric way. In the linear case it takes the form:

$$\text{HSIC}_{lin}(X, Y) = \frac{1}{(n-1)^2} tr(XX^\top YY^\top) = \left\| cov(X^\top, Y^\top) \right\|_F^2 \tag{1}$$

Where subscript F denotes the Frobenius norm of a matrix. Linear CKA can then be computed as:

$$\text{CKA}_{lin}(X, Y) = \frac{\text{HSIC}_{lin}(X, Y)}{\sqrt{\text{HSIC}_{lin}(X, X)\text{HSIC}_{lin}(Y, Y)}} \tag{2}$$

Intuitively, HSIC computes the similarity structures of X and Y through the Gram matrices of internal products ($XX^\top$ and $YY^\top$) and then compares these similarity structures by computing their alignment through the trace of $XX^\top YY^\top$.

3 Experiments and Results

3.1 CKA Early Layer Results

CKA values are often treated as a surrogate metric to measure the usefulness and similarity of a network's learned features when compared to another network (Ramasesh et al., 2021). In order to analyze this common assumption, we compare the features of: (1) a network trained to generalize on the CIFAR10 image classification task (Krizhevsky et al., 2009), (2) a network trained to "memorize" the CIFAR10 images (i.e. permuted target labels), and (3) an untrained randomly initialized network (for network architecture and training details see the Appendix). As show in Fig. 1, early layers of these networks should have very similar representations given the high CKA values. Under the previously presented assumption, one should therefore conclude that the learned features at these layers are relatively similar and equally valuable. However this is not the case, we can see in Fig. 2 that the convolution filters are drastically different across the three networks. Moreover, Fig. 2 elucidates that considerably high CKA similarity values for early layers, does not necessarily translate to more useful, or similar, captured features.

3.2 Sensitivity to affine transformations

We consider artificially generated representations $X \in \mathbb{R}^{n \times d}$ to which we apply affine transformations to obtain $Y \in \mathbb{R}^{n \times d}$. We generate X by sampling 10K points from the 1K-dimensional unit cube centered at the origin and 10K points from a similar cube centered at $(1.1, 0, 0, \ldots, 0)$ so the points from the two cubes are linearly separable along the first dimension.

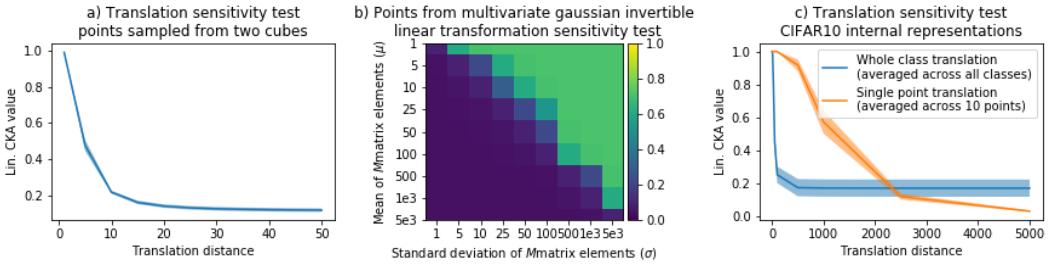

Figure 3: Linear CKA value between the artificial representations X and either **a)** the translated version Y as a function of the translation distance c or **b)** $Y = XM$ with M being an invertible matrix with elements sampled from $\mathcal{N}(\mu, \sigma^2)$ as a function of μ and σ. The mean and standard deviation across 10 random instantiations of v the translation direction and M are shown. **c)** CKA value between a CNN's internal representations of the CIFAR10 training set and modified versions where either a class is translated or a single point is translated as functions of the translation distance.

The first transformation we consider is translating the subset of representations sampled from the second cube by $c \in \mathbb{R}$ in direction $\mathbf{v} \in \mathbb{R}^d$ sampled from the the d-dimensional ball, we plot the CKA values between X and Y as a function of c in Fig. 3a). This transformation entirely preserves the topological structure of the representations as well as their local geometry since the points sampled from each cube have not moved with respect to the other points sampled from the same cube and the two cubes are still separated, only the distance between them has been changed. Despite these multiple notions of "similarity" between X and Y being preserved, the CKA values quickly drop below 0.2 even for relatively small translation distances.

The second transformation considered is a multiplication by a matrix $M \in \mathbb{R}^{d \times d}$ whose elements are sampled from a Gaussian with mean μ and standard deviation σ. We verify the invertibility of M since it is not guaranteed and only keep invertible matrices. We show the CKA values between X and the transformed Y in Fig. 3b). Since this is a invertible linear transformation we would expect it to only modestly change the representations in X and the CKA value to be only slightly lower than 1. However, we observe that even for small values of μ and σ, CKA drops to 0, which would indicate that the two sets of representations are dissimilar and not linked by a simple transformation.

3.3 Constructing a low CKA similarity

We further experiment with CKA's sensitivity to translations but in a more realistic setting, while preserving important semantic information. We consider the 9 layers CNN presented in Section 6.1 of Kornblith et al. (2019) trained on CIFAR10. When trained on classification tasks, ANNs tend to learn increasingly complex representations of the input data that can be more or less linearly separated into classes by the last layer of the network. With this in mind, we argue that a relevant way in which two sets of representations can be "similar" in practice is if they are linearly separable by the same hyperplanes in parameter space, the same linear classifier can then correctly classify both sets of representations. Given X, the network's internal representations of 10k training images at the last layer before the output, we can create Y by translating examples corresponding to a certain class in a way that does not affect the linear separability of the representations. We use an SVM classifier to extract the hyperplanes in parameter space, which best separate the data (with approx. 91% success rate). We then translate a subset of the representations in a direction which will never cross these hyperplanes. We plot the CKA values between X and Y according to the translation distance in Fig. 3c). The CKA values quickly drop to 0, despite the existence of a linear classifier that can classify both sets of representations into the correct classes with $> 90\%$ accuracy.

In Fig. 3c) we also examine linear CKA's sensitivity to outliers. Plotted are the CKA values between the set of training set representations and the same representations but with a single point being translated far from its original location. While the translation distance needed to achieve low CKA values is relatively high, the fact that the position of a *single* point out of *tens of thousands* can so drastically influence the CKA value raises doubts about CKA's effectiveness as a similarity metric.

3.4 Optimizing CKA Map

The CKA map, commonly used to analyze network architectures (Kornblith et al., 2019) and their inter-layers similarities, is a matrix M, where $M[i, j]$ is the CKA value between the activations of layers i, and j of a network. We set to directly manipulate the CKA map of a trained network f_{θ^*}, by adding the desired CKA map to its optimization objective, while maintaining its original outputs via distillation loss (Hinton et al., 2015).

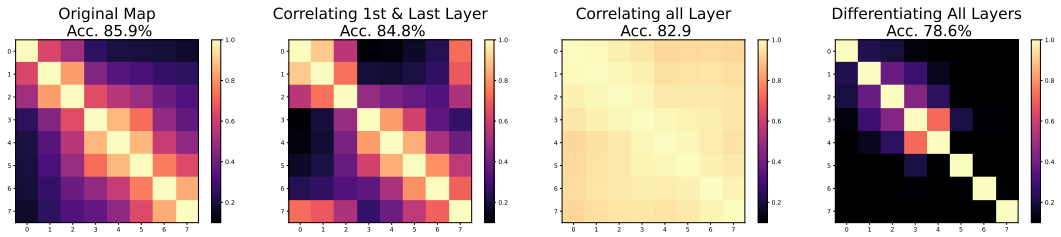

Figure 4: Original Map is the CKA map of a network trained on CIFAR10. We manipulate this network to produce CKA maps which: (1) maximizes the CKA similarity between the 1st and last layer, (2) maximizes the CKA similarity between all layers, and (3) minimizes the CKA similarity between all layers. In cases (1) and (2), the network experiences only a slight loss in performance, which counters previous findings by achieving a strong CKA similarity between early and late layers.

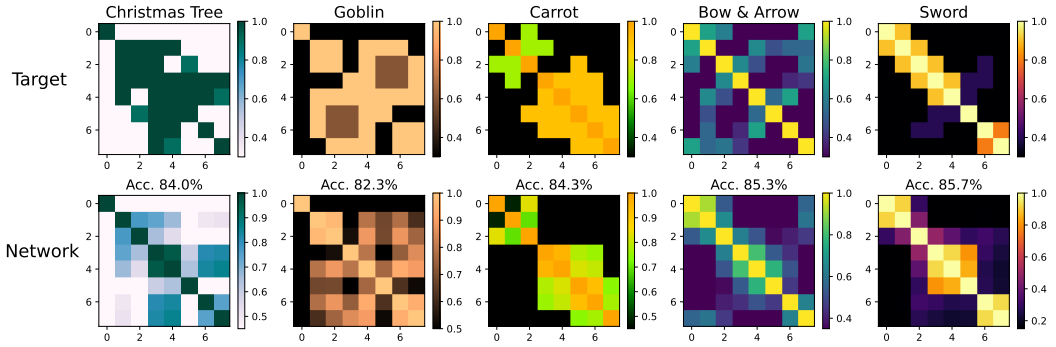

Figure 5: The comical target CKA maps (first row) are used as the objective for the CKA map loss in Eq. 3. The second row shows the CKA map produced by the network.

We further optimize f_{θ^*} over the training set (X, Y) following the equation:

$$\theta^*_{new} = \arg\min_{\theta} \left(\mathcal{L}_{distill} \left(f_{\theta^*}(X), f_\theta(X) \right) + \lambda \mathcal{L}_{map} \left(M_{f_\theta(X)}, M_{target} \right) \right) \tag{3}$$

Where $\mathcal{L}_{map} = \ln\cosh\left(M_{f_\theta(X)} - M_{target}\right)$. Fig. 4 shows the CKA map of f_{θ^*} along with the CKA map of three scenarios we investigated: (1) maximizing the CKA similarity between the 1st and last layer, (2) maximizing the CKA similarity between all layers, and (3) minimizing the CKA similarity between all layers (for network architecture and training details see Appendix). In cases (1) and (2), the network performance is barely hindered by the manipulations of its CKA map. This is surprising and contradictory to the previous findings (Kornblith et al., 2019; Raghu et al., 2021) as it suggests that it is possible to achieve a strong CKA similarity between early and later layers of a well-trained network. We further manipulated the CKA map of f_{θ^*} to produce a series of comical CKA maps (Fig. 5). Although the network CKA maps seen in Fig. 5 closely resemble their respective targets, it should be noted that we prioritized maintaining the network outputs, and ultimately its accuracy by choosing small λ values in Eq. 3. Higher values of λ results in stronger agreements between the target and network CKA maps at the cost of performance.

4 Conclusion

We continue a very recent line of research by showing situations in which linear CKA is inadequate as a representation similarity measure. Linear CKA attributes low similarity to sets of representations that are directly linked by simple affine transformations preserving important functional characteristics and it attributes high similarity to representations from very different networks. Furthermore, we can manipulate CKA to give arbitrarily low/high values while preserving functional behaviour. An important direction of improvement for our work is to theoretically study CKA's sensitivity to some of the used transformations. We hope our work inspires ML researchers to also consider other similarity measures and potentially different data analysis and visualization algorithms when studying neural network representations as opposed to relying solely on CKA to draw conclusions. It would also be interesting to see whether or not results obtained using other methods of analysis would support scientific conclusions already established using linear CKA.

REFERENCES

Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. Grounding representation similarity through statistical testing. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=_kwj6V53ZqB.

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Lecture Notes in Computer Science*, pp. 63–77. Springer Berlin Heidelberg, 2005. doi: 10.1007/11564089_7. URL https://doi.org/10.1007/11564089_7.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pp. 3519–3529. PMLR, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, MIT & NYU, 2009.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/a7a3d70c6d17a73140918996d03c014f-Paper.pdf.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 512–523. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/0607f4c705595b911a4f3e7a127b44e0-Paper.pdf.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KJNcAkY8tY4.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=LhY8QdUGSuw.

Alex H Williams, Erin Kunz, Simon Kornblith, and Scott Linderman. Generalized shape metrics on neural representations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=L9JM-pxQOl.

5 Appendix

5.1 Network Architecture

In Sec. 3.1 we use a 9 layer neural network; the first 8 of these layers are convolution layers and the last layer is a fully connected layer used for classification. We use ReLU (Nair & Hinton, 2010) throughout the network. The kernel size of every convolution layer is set to $(3, 3)$ except the first two convolution layers, which have $(7, 7)$ kernels. All convolution layers follow a padding of 0 and a stride of 1. Number of kernels in each layer of the network, from the lower layers onward follows: $[16, 16, 32, 32, 32, 64, 64]$. In this network, every convolution layer is followed by batch normalization (Ioffe & Szegedy, 2015). The network we used in Sec. 3.4 is similar to the network we just described, except the kernel size for all layers are set to $(3, 3)$.

5.2 Training Details

The models in Sec. 3.1, both the generalized and memorized network, were trained for 100 epochs using AdamW (Loshchilov & Hutter, 2017) optimizer with a learning rate (LR) of $1e{-}3$ and a weight decay of $5e{-}4$. The LR is follows cosine LR scheduler (Loshchilov & Hutter, 2016) with an initial LR stated earlier.

The training of the base model (original) model in Sec. 3.4 follows the same training procedure as of the models from Sec. 3.1, except in this setting we train the model for 200 epochs, with an initial LR of 0.01. All other models in Sec. 3.4 (with a target CKA map to optimize) are also trained with similar training hyperparameters to that of the base model, except the followings: (1) these models are only trained for 50 epochs. (2) the objective function includes a hyperparameter λ (see Eq. 3), which we set to 100 for all models. (3) The cosine LR scheduler includes a warm-up step of 500 optimization steps. (4) the LR is set to $1e{-}3$.